Scope-bounded Multistack Pushdown Systems: Fixed-Point, Sequentialization, and Tree-Width*

Salvatore La Torre[1] **and Gennaro Parlato**[2]

1 **Dipartimento di Informatica,**
Università degli Studi di Salerno, Italy
slatorre@unisa.it
2 **School of Electronics and Computer Science**
University of Southampton, UK
gennaro@ecs.soton.ac.uk

Abstract

We present a novel fixed-point algorithm to solve reachability of multi-stack pushdown systems restricted to runs where matching push and pop transitions happen within a bounded number of context switches. The followed approach is compositional, in the sense that the runs of the system are summarized by bounded-size interfaces. Moreover, it is suitable for a direct implementation and can be exploited to prove two new results. We give a sequentialization for this class of systems, i.e., for each such multi-stack pushdown system we construct an equivalent single-stack pushdown system that faithfully simulates the behavior of each thread. We prove that the behavior graphs (multiply nested words) for these systems have bounded tree-width, and thus a number of decidability results can be derived from Courcelle's theorem.

1998 ACM Subject Classification D.2.4 Software/Program Verification.

Keywords and phrases Model-checking, multi-threaded programs, sequentialization, tree-width.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.173

1 Introduction

Multi-stack pushdown systems (MPDS) accurately capture the control-flow of concurrent programs communicating via shared memory, and thus, are widely used as an abstract model of such programs in several analysis problems, such as reachability and more in general model-checking.

It is well known that MPDS with two stacks can simulate Turing machines. A recent line of research has concerned with decidable syntactic restrictions that limit the behaviors of the general model, such as bounding the number of context-switches [21] or the number of phases [9, 10] in a run.

Last year in [15], a new restriction that limits the *scope* of matching push and pop transitions in terms of number of context switches (*scope-bounded restriction*) has been considered. With this limitation, the analysis is carried over only the system executions where each symbol pushed onto a stack is either popped within a bounded number of context switches or is never popped. As a matter of fact, the scope-bounded restriction is strictly more permissive than bounded-context switching, in fact it allows us to account for computations with unboundedly many contexts, and thus, with an unbounded number of interactions between the threads (see [15]).

* This work was partially funded by the FARB grants 2010-2012 Università degli Studi di Salerno (Italy).

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 173–184

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Also, in [15] the reachability of MPDS under the scope-bounded restriction is shown to be PSPACE-complete. For an n stack MPDS, the given decision algorithm characterizes the configurations which are k-scoped reachable by computing tuples which store: (1) the reached control state, (2) the (top) portion of the stack contents of all the symbols that have been pushed in the last k rounds, and (3) the control states at the context-switches in the last round (computations are arranged in rounds of thread executions where each thread is activated exactly once). The stack contents are summarized as finite automata of a fixed form: each automaton has as states k copies of the control states along with an initial state, and differ from each other only in the transitions.

In this paper, we adopt the same restriction on the MPDS computations and contribute to this research in several ways.

As a first result, we develop a new algorithm to solve the scope-bounded reachability problem for MPDS. The solution we propose is fixed-point and uses the concept of interface of a thread computation introduced in [13]. A *thread-interface* simply summarizes the executions of a thread in consecutive rounds of executions of a system computation, by storing the control states of the starting and ending configurations in each round. An interesting feature of thread-interfaces is that they can be used compositionally to summarize entire runs. A key result that we prove here and exploit to design our fixed-point algorithm is that it is sufficient to store n-tuples of fragments of thread-interfaces over at most k rounds, to reconstruct the summaries of entire k-scoped runs of an MPDS with n stacks.

The proposed algorithm have a simpler formulation than that given in [15] and seems to be more suitable for efficient implementation. Thread interfaces are a simpler artifact than finite automata and can be easily encoded for efficient symbolic search. In fact, our fixed-point algorithm has a direct implementation in the tool GETAFIX, a framework that supports the writing in a fixed-point calculus of model-checkers for sequential and concurrent Boolean programs (see [11]). Moreover, dealing with simpler tuples seems to avoid some redundancy. In fact, if on the one side computing such automata is computationally equivalent to compute thread interfaces, on the other side our algorithm searches over essentially Q^{2kn} tuples while the previous one over $Q^n 2^{n(k^2Q^2+\mu)}$ tuples, where Q denotes the set of control states and μ denotes subquadratic terms in k and Q.

The approach followed in our above fixed point algorithm can be used in two directions to obtain interesting results which constitute the other contributions of this paper.

First, the fixed-point rules used to compose, accumulate and simplify the thread-interfaces in our algorithm can be re-used to construct a single-stack pushdown system that simulates the k-scoped runs of an n-stack MPDS. For computer programs, this corresponds to a *sequentialization*, i.e., a transformation of a concurrent program into an *equivalent* sequential one. Sequentializations have recently received great attention in the context of program verification with the goal of performing the analysis of concurrent programs via tools designed for sequential ones (e.g., see [18, 5]). Several tools have been developed on this paradigm and have allowed to find bugs that could not be found with other approaches [8], or even prove programs entirely correct [13].

Second, we show that multiply nested words, which allow us to represent runs of MPDS with graphs, have a bounded tree-width when restricting to bounded scope. Again, the executions of the fixed point algorithm are the key concept of this proof. Moreover, since this class of multiply nested words can be captured in the MSO logic, we can inherit all the decidability results of [20]. In particular, all properties that can be expressed in MSO can be shown decidable using this result, including the decidability of linear temporal logic.

Related work. Besides the already cited research there are a few other works which

are related to ours. We start mentioning some recent results that have concerned MPDS restricted to scope-bounded computations that have followed the results presented in this paper. In [7], the authors show that bounded-scoped multiply nested words have bounded tree-width using the notion of split-width there introduced. In [1], the model-checking of scope-bounded MPDS is shown to be EXPTIME-complete for linear-time temporal logic (LTL). Independently, in [16], a logic for multiply nested words which extends LTL and allows to capture the call-return relations within each tread (MultiCaRet) is introduced and, among other results, the related model-checking and satisfiability problems are shown to be EXPTIME-complete when restricting to scope-bounded computations.

The notion of bounded-context switching has been successfully used for: model-checking tools for concurrent Boolean programs [11, 18, 22] and Boolean abstractions of parameterized programs [13]; sequentializations of shared-memory concurrent programs [12, 18] and their use with SMT solvers to find errors in concurrent programs [8];sequentialization of multiprocessor programs communicating through asynchronous message passing [4]; sequentializations of shared-memory parameterized programs [14]; translation of concurrent programs under total store ordering memory model to concurrent programs under sequential consistency memory model [2]; model-checking of programs with dynamic thread creation [3]; analysis of systems with heaps [6], and weighted pushdown systems [19].

2 Multistack Pushdown Systems

Given two positive integers i and j, $i \leq j$, we denote with $[i, j]$ the set of integers k with $i \leq k \leq j$, and with $[j]$ the set $[1, j]$.

A multi-stack pushdown system consists of a finite number of pushdown automata each of which with its local stack, that communicate through the shared control states. Multi-stack pushdown system is a faithful model to represent concurrent Boolean programs, where each pushdown component models a single thread and the shared control states can be used to allow shared communication among them.

▶ **Definition 1.** (MULTI-STACK PUSHDOWN SYSTEMS) Let $n \in \mathbb{N}$. A *n-stack pushdown system* (n-MPDS) is a tuple $M = (Q, q_0, \Gamma, \{(\delta_i^{int}, \delta_i^{push}, \delta_i^{pop})\}_{i\in[n]})$ where Q is a finite set of control states, $q_0 \in Q$ is the initial state, Γ is a finite stack alphabet, and for every $i \in [n]$, $\delta_i^{int} \subseteq (Q \times Q)$ is a set of internal transitions and $\delta_i^{push}, \delta_i^{pop} \subseteq (Q \times \Gamma \times Q)$ are respectively push and pop transitions involving the i'th stack. For every $i \in [n]$, with M_i we denote the i'th thread of M, i.e., the 1-MPDS $(Q, q_0, \Gamma, \{(\delta_i^{int}, \delta_i^{push}, \delta_i^{pop})\})$. ◀

An M *configuration* is a tuple $C = \langle q, \{w_i\}_{i\in[n]}\rangle$, where $q \in Q$ and each $w_i \in \Gamma^*$ is the content of the i'th stack. Moreover, C is *initial* if $q = q_0$ and $w_i = \varepsilon$, for every $i \in [n]$. Let $Act_i = \{int_i, push_i, pop_i\}$ be the set of *actions* of thread M_i, and $Act = \bigcup_{i\in[n]} Act_i$ be the set of all *actions* of M. A transition between two configurations over an action $\sigma \in Act$ is defined as follows:
$\langle q, \{w_i\}_{i\in[n]}\rangle \xrightarrow{\sigma}_M \langle q', \{w_i'\}_{i\in[n]}\rangle$ if one of the following holds for some $i \in [n]$

[Internal] $\sigma = int_i$, $(q, q') \in \delta_i^{int}$, and $w_h' = w_h$ for every $h \in [n]$.
[Push] $\sigma = push_i$, $(q, \gamma, q') \in \delta_{push}^i$, $w_i' = \gamma.w_i$, and $w_h' = w_h$ for $h \in ([n] \setminus \{i\})$.
[Pop] $\sigma = pop_i$, $(q, \gamma, q') \in \delta_{pop}^i$, $w_i = \gamma.w_i'$, and $w_h' = w_h$ for $h \in ([n] \setminus \{i\})$.

A *run* ρ of M from C_0 to C_m, with $m \geq 0$, denoted $C_0 \rightsquigarrow_M C_m$, is a possibly empty sequence of transitions $C_{j-1} \xrightarrow{\sigma_j}_M C_j$, for every $j \in [m]$. Furthermore, ρ is a *computation* of M if C_0 is initial.

Scope-bounded Runs. Let $\rho = C_0 \xrightarrow{\sigma_1} C_1 \xrightarrow{\sigma_2} \dots C_{m-1} \xrightarrow{\sigma_m}_M C_m$ be a run of an n-MPDS M. We associate to each transition in ρ a *round number*. The map $round^\rho : [m] \to \mathbb{N}$ is inductively defined as follows:

$$round^\rho(r) = \begin{cases} 1 & \text{if } r = 1; \\ round^\rho(r-1) + 1 & \text{if } (r > 1) \ \&\ (\sigma_{r-1} \in Act_i) \\ & \quad \&\ (\sigma_r \in Act_{i'}) \ \&\ (i > i'); \\ round^\rho(r-1) & \text{otherwise.} \end{cases}$$

For any M run ρ and $i \in [n]$, $\mu_i^\rho(s,t)$ is the predicate that holds true whenever the t'th transition of ρ pops the symbol pushed on stack i at the s'th transition. Formally, μ_i^ρ is the unique predicate over the set $[m]^2$ such that: if $\mu_i^\rho(s,t)$ holds true then $s < t$, $\sigma_s = push_i$, $\sigma_t = pop_i$, and

- for every $t' \in [m]$ with $s < t' < t$, if $\sigma_{t'} = pop_i$ then there is an index $s' \in [m]$ with $s < s' < t'$ such that $\sigma_{s'} = push_i$ and $\mu_i^\rho(s',t')$ holds true;
- for every $s' \in [m]$ with $s < s' < t$, if $\sigma_{s'} = push_i$ then there is an index $t' \in [m]$ with $s' < t' < t$ such that $\sigma_{t'} = pop_i$ and $\mu_i^\rho(s',t')$ holds true.

Furthermore, if $\sigma_t = pop_i$ then there is an s such that $\mu_i^\rho(s,t)$ holds true.

For $k \in \mathbb{N}$, ρ is *k-scoped* iff for every $i \in [n]$ and two indices $s,t \in [m]$ if $\mu_i^\rho(s,t)$ holds true then $(round^\rho(t) - round^\rho(s)) < k$. In other words, in any k-scoped run any pop operation retrieves from the stack a symbol that has been pushed within the last k rounds.

To illustrate the above concepts consider the run of a 2-MPDS in Figure 1. To simplify the representation, we have omitted the stack contents and reported only the control state of the configurations. Also, we have repeated the control location ending a row at the beginning of the following one such that each row corresponds to a whole round. The states where the control switches from the first to the second thread in each round have been aligned under a column. To emphasize the matching of push and pop we use a different alphabet letter for denoting each push and the same letter with a bar for the matching pop. We use subscripts to distinguish among the different threads.

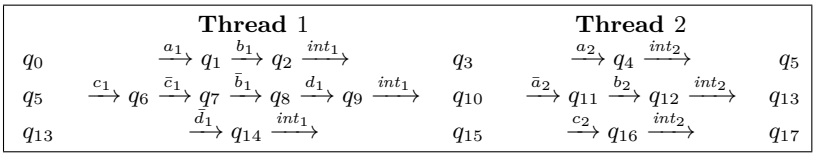

Figure 1 A sample 3-round run of a 2-MPDS.

The run of Figure 1 is k-scoped for any $k \geq 2$ since matching push/pop spans over at most 2 rounds, and it is not 1-scoped, in fact for example the push denoted b_1 in round 1 is matched in round 2.

3 Interfaces

In this section, we introduce the concept of *thread-interface* which is central for the paper. Informally, a thread-interface summarizes for a single thread the computation within consecutive rounds, and under some conditions, can be composed with the thread-interfaces of the other threads to summarize entire runs of an MPDS. We show that when restricting to k-scoped runs, the whole computation of a single thread across unboundedly many rounds can be indeed captured by composing thread-interfaces over at most k rounds. This will be exploited in the next section to give a fixed-point algorithm to solve the reachability problem restricted to k-scoped runs of an MPDS.

▶ **Definition 2.** (THREAD-INTERFACE) Let $M = (Q, q_0, \Gamma, \{(\delta_i^{int}, \delta_i^{push}, \delta_i^{pop})\}_{i\in[n]})$ be an n-MPDS. For each $i \in [n]$, an *i-thread-interface* of M is a possibly empty tuple $I = \langle in_j, out_j\rangle_{j\in[r]}$, for some $r \in \mathbb{N}$ (the *dimension* of I, also denoted $\dim(I)$), such that if $r > 0$ there exist r runs $\pi_1, \pi_2, \ldots, \pi_r$ of M_i in which

- for every $j \in [r]$, $\pi_j = \langle in_j, w_j\rangle \rightsquigarrow_{M_i} \langle out_j, w'_j\rangle$ is a run of M_i;
- $w_1 = \epsilon$, and for every $j \in [r-1]$, $w_{j+1} = w'_j$. ◀

Fix for the rest of the section a 2-MPDS P with a run as in Figure 1. From the above definition the tuple $T_1 = (\langle q_0, q_3\rangle, \langle q_5, q_{10}\rangle, \langle q_{13}, q_{15}\rangle)$ is a 1-thread-interface of P of dimension 3 and $T_2 = (\langle q_3, q_5\rangle, \langle q_{10}, q_{13}\rangle, \langle q_{15}, q_{17}\rangle)$ is a 2-thread-interface of P of dimension 3. Note that since a run has possibly zero transitions, also $T_3 = (\langle q_0, q_3\rangle, \langle q_5, q_5\rangle, \langle q_5, q_{10}\rangle, \langle q_{13}, q_{15}\rangle)$ and $T_4 = (\langle q_3, q_5\rangle, \langle q_{10}, q_{13}\rangle, \langle q_{15}, q_{15}\rangle)$ are thread-interfaces.

For $h = 1, 2$, let $I_h = \langle in_j^h, out_j^h\rangle_{j\in[r_h]}$ be an i-thread-interface of M, for some $i \in [n]$. We define two internal operations over thread-interfaces of a given thread. With $I_1 \bowtie_1 I_2$ we denote the tuple obtained by appending I_2 to I_1. Formally, $I_1 \bowtie_1 I_2 = \langle in_j, out_j\rangle_{j\in[r_1+r_2]}$ where $in_j = in_j^1$ and $out_j = out_j^1$ for $j \in [r_1]$, and $in_{r_1+j} = in_j^2$ and $out_{r_1+j} = out_j^2$ for $j \in [r_2]$. The other operation is a variation of $\bowtie_1$ where the last pair of I_1 is composed with the first pair of I_2. It is defined when I_1 and I_2 are both not empty. Formally, if $r_1, r_2 > 0$ and $out_{r_1}^1 = in_1^2$, then we denote with $I_1 \bowtie_2 I_2$ the tuple $\langle in_j, out_j\rangle_{j\in[r_1+r_2-1]}$ where $in_j = in_j^1$ and $out_j = out_j^1$ for $j \in [r_1 - 1]$, $in_{r_1} = in_{r_1}^1$, $out_{r_1} = out_1^2$, and $in_{r_1+j} = in_{j+1}^2$ and $out_{r_1+j} = out_{j+1}^2$ for $j \in [r_2 - 1]$.

Directly from the definition of thread-interface we get that both compositions define thread interfaces.

▶ **Lemma 3.** *Let $I_h = \langle in_j^h, out_j^h\rangle_{j\in[r_h]}$ be a i-thread-interface of M, for some $i \in [n]$ and $h = 1, 2$.*

$I_1 \bowtie_1 I_2$ is a i-thread-interface of dimension $r_1 + r_2$.

If $out_{r_1}^1 = in_1^2$, then $I_1 \bowtie_2 I_2$ is a i-thread-interface of dimension $r_1 + r_2 - 1$.

The two compositions $\bowtie_1$ and $\bowtie_2$ are sufficient to fully characterize, by thread-interfaces of bounded dimension all the thread-interfaces "canonically" defined by scope-bounded runs of an MPDS. Given an m-round run ρ of an n-MPDS M, a i-thread-interface $I = \langle in_j, out_j\rangle_{j\in[m]}$ is *canonical for ρ* if along ρ for each round j the computation of thread M_i starts at in_j and ends at out_j. The idea is thus to capture with each i-thread-interface a portion ρ' of the run, where all the occurrences of pushes over the i'th stack are either matched within ρ' or are never matched in the whole run. Due to the k-scoped restriction, for all matched pushes the matching pop transition must occur within k rounds, and since the matching pairs of push/pops define a nested relation, each such portion ρ' can be taken such that it spans over at most k rounds.

As an example, consider again the 2-scoped run from Figure 1. Note that T_1 and T_2 are canonical thread-interfaces for it, and $T_1 = (\langle q_0, q_3\rangle, \langle q_5, q_8\rangle) \bowtie_2 (\langle q_8, q_{10}\rangle, \langle q_{13}, q_{15}\rangle)$ and $T_2 = (\langle q_3, q_5\rangle, \langle q_{10}, q_{13}\rangle) \bowtie_1 (\langle q_{15}, q_{17}\rangle)$ (the interfaces used in the compositions are all of dimension at most 2).

The above result is formally stated in the following lemma.

▶ **Lemma 4.** *Let $k \in \mathbb{N}$, M be an n-MPDS, ρ be a k-scoped run of M, and I be a canonical i-thread-interface for ρ, $i \in [n]$.*

There exist i-thread-interfaces $I_0, \ldots, I_s$ of dimension at most k such that $I = I_0 \bowtie_{j_1} I_1 \ldots \bowtie_{j_s} I_s$ with $j_1, \ldots j_s \in [2]$.

For i=1,2, let $I_i = \langle in_j^i, out_j^i \rangle_{j \in [r_i]}$ be a thread-interface of M. We say that I_1 *stitches to* I_2 *up to index* r (shortly, r-stitches) if $r \leq \min\{r_1, r_2\}$, and $out_j^1 = in_j^2$ for every $j \in [r]$. Also, I_2 *wraps with* I_1 *up to index* r (shortly, r-wraps) if $r \leq \min\{r_2, r_1 - 1\}$ and $out_j^2 = in_{j+1}^1$ for every $j \in [r]$. Note that, in particular, if either I_1 or I_2 is empty (i.e., dimension is 0), I_1 does not r-stitch to I_2 and I_1 does not r-wrap with I_2 for any $r > 0$.

In our running example, T_1 3-stitches to T_2 and T_2 2-wraps with T_1, and the two interfaces correspond to the run in Figure 1, and similarly the pair T_1, T_4 corresponds to the run portion from q_0 through q_{15}.

In general, we can show that runs of MPDS can be fully characterized by tuples of thread-interfaces. In fact, by definition, each m-round run of an n-MPDS M exactly defines a canonical thread-interface I_i where $i \in [n]$ such I_i m-stitches to I_{i+1}, for every $i \in [n-1]$, and I_n $(m-1)$-wraps with I_1. Vice-versa given the i-thread-interfaces I_i with $\dim(I_i) = m$, $i \in [n]$, such that I_i m-stitches to I_{i+1}, for every $i \in [n-1]$, and I_n $(r-1)$-wraps with I_1, from the definition of thread-interface we can construct an m-rounds run of M by concatenating the runs corresponding to each interface. Also, observe that $I_1, \ldots, I_n$ are the canonical thread-interfaces of the constructed run, and for $j \in [m]$, their j'th pairs contain the states at which the constructed run context-switches in round j. Thus, we get the following lemma.

▶ **Lemma 5.** *Let M be an n-MPDS, and q be an M control state. Then, there is a run of M reaching q iff there are n thread-interfaces $I_1, I_2, \ldots, I_n$ all of dimension r, where $I_i = \langle in_j^i, out_j^i \rangle_{j \in [r]}$ is a i-thread-interface of M, such that*

- *I_i r-stitches to I_{i+1}, for every $i \in [n-1]$;*
- *I_n $(r-1)$-wraps with I_1;*
- *in_1^1, the first input state of I_1, is the initial state of M and*
- *out_r^n, the last output state of I_n, is q.*

4 Fixed-Point Algorithm for Scope-Bounded Reachability

In this section, we describe a new algorithm to solve the scope-bounded reachability problem for MPDS. We recall that this problem has been recently shown to be decidable in [15]. Besides the differences in the approach, the solution we present here is fixed-point and has several advantages. First, our algorithm has a direct implementation in the tool GETAFIX [11]. Moreover, from our fixed-point characterization, we can easily derive a straightforward sequentialization algorithm, as well as prove that multiply nested words of scope-bounded runs have bounded tree-width, and therefore, a number of properties (including scope-bounded reachability) can be shown to be decidable by Courcelle's theorem (see Section 5). We start defining the scope-bounded reachability problem.

▶ **Definition 6** (SCOPE-BOUNDED REACHABILITY PROBLEM)**.** Fix $k \in \mathbb{N}$. For an n-MPDS M and an M control state q, the *k-scoped reachability problem* asks whether there is a k-scoped run of M from an initial configuration to any configuration of the form $\langle q, \{w_i\}_{i \in [n]} \rangle$. ◀

The algorithm. One way to solve the scope-bounded reachability problem is to first non-deterministically compute n thread-interfaces of the same dimension, one for each thread, and then by Lemma 5 check whether they form an M computation reaching state q. The drawback of this approach is that it gives a semi-algorithm as we do not know, a priori, the number of rounds that would be needed to conclude that q is not reachable. In contrast, the solution we propose, as a fixed-point algorithm, would implement the same approach as outlined above with the difference that we do not generate thread interfaces one after another, but rather in parallel, as the components of a tuple.

At each step, we append via the operators $\bowtie_1$ and $\bowtie_2$ a thread-interface to the component of the tuple that has the least dimension (*thread-interface progression rule*). In doing so, we also check that the appended thread-interface is compatible with the rest of the tuple (i.e., it appropriately stitches to its neighbours in the tuple and for the first/last component it wraps with the opposite end-tuple).

By computing the canonical thread-interfaces this way, as soon as the pairs of states of index j have been added to all components of the tuple, all of them can be safely removed (provided that if the first component does not have the pair of index $j+1$, we store the *out*-state removed from the last component). This corresponds to advancing the starting round of the tuple to the next round in a run matching the tuple of canonical thread-interfaces (*round deletion rule*).

To minimise the size of the components in the stored tuples, we apply the round deletion rule with higher priority than the thread-interface progression rule. This way, we can keep the dimension of each tuple component not larger than k, and this ensures also the convergence of the algorithm.

Our algorithm maintains tuples of the form $\nu = [I_1, \ldots, I_n]$ where each I_i is a *fragment* of an i-thread-interface: if ν is computed by our algorithm, then there exist n canonical thread-interfaces $I'_1, \ldots, I'_n$, where I'_i is an i-thread interface, which satisfy the conditions of Lemma 5, and furthermore, $I'_i \bowtie_1 I_i$ is an i-thread-interface. Note that each I_i may not be a thread-interface.

We implement the above mentioned priority by firing the thread-interface progression rule only on tuples containing a *basic* fragment of thread interface. For a tuple $\nu = [I_1, \ldots, I_n]$, I_j is *basic* if it is empty, or has dimension 1 and does not match completely the corresponding execution context in ρ (in particular, it matches the state when context-switching into such context but does not match the state when context-switching out of it). Formally, we say that I_j, $j \in [n]$, is *basic for* $[I_1, \ldots, I_n]$ if either one of the following conditions holds:

(*i*) $I_j = \emptyset$, *or*

(*ii*) $j < n$, $\dim(I_j) = 1$, $I_{j+1} \neq \emptyset$, and I_j does not 1-stitch to I_{j+1}, *or*

(*iii*) $j = n$, $\dim(I_n) = 1$, $\dim(I_1) > 1$, and I_n does not 1-wrap with I_1.

A thread-interface $I = \langle in_j, out_j \rangle_{j \in [r]}$ is *initial* if $r > 0$ and $in_1 = q_0$. Denoting with $\emptyset$ the empty thread-interface, the set of tuples computed by the algorithm, denoted $\mathcal{I}$, is initialized to all n-tuples $[I, \emptyset, \ldots, \emptyset]$ where I is an initial 1-thread-interface.

The detailed rules of the algorithm are given in Figure 2. There, we have denoted with $\bowtie$ the extension of $\bowtie_2$ such that $I \bowtie J$ is J, if $I = \emptyset$, and $I \bowtie_2 J$, otherwise. Note that $\bowtie$ is defined as $\bowtie_1$ when the first argument is $\emptyset$, and thus captures the composition of thread-interfaces via $\bowtie_1$ in the thread-interface progression rule. Also, in the thread-interface progression rule we do not force the matching on the last index value for J. This is to capture the cases when the composition of the canonical thread-interface requires the use of the $\bowtie_2$ operator. Finally, observe that for tuples where the 1-thread-interface has dimension 1, with the round deletion rule we do not simply delete this thread-interface but we replace it with the 1-thread-interface $(\langle out_1, out_1 \rangle)$ where out_1 is the out-state of the first pair of the last fragment in the tuple. The reason we handle the first thread differently from the others resides in the fact that the matching of state out_1 with the corresponding in-state of the 1-thread-interface (wrapping condition) cannot be checked at this time since this thread-interface has dimension 1. Therefore, it is necessary to store it for future matching.

The thread-interfaces of dimension at most k can be computed in a standard way, see for example [11], and thus we omit it. The algorithm halts when no more tuples can be added to the set $\mathcal{I}$.

Initialization.
$\mathcal{I} = \{[I, \emptyset, \ldots, \emptyset] \mid I \text{ is an initial 1-thread-interface of } M \text{ with } \dim(I) \leq k\}$

Thread-interface progression—append thread-interface to basic thread-interface fragment.
`Let` $[I_1, \ldots, I_n] \in \mathcal{I}$ and for some $j \in [n]$, I_j be basic for $[I_1, \ldots, I_n]$.
`Add` to $\mathcal{I}$ any tuple obtained from $[I_1, \ldots, I_n]$ by replacing I_j with $J = I_j \bowtie J'$ such that:
1. J' is a j-thread-interface of M of dimension $s \leq k$,
2. (For every $h \in [n]$, let r_h be the dimension of I_h)
   - `if` $j = 1$ `then` I_n r-wraps with J for $r = \min\{s, r_n - 1\}$
   - `if` $j > 1$ `then` I_{j-1} r-stitches to J for $r = \min\{s, r_{j-1} - 1\}$
   - `if` $j < n$ `then` J r-stitches to I_{j+1} for $r = \min\{s - 1, r_{j+1}\}$
   - `if` $j = n$ `then` J r-wraps with I_1 for $r = \min\{s - 1, r_1\}$

Round deletion—remove completed rounds from tuples.
`Let` $[I_1, \ldots, I_n] \in \mathcal{I}$ be s. t. I_j 1-stitches to I_{j+1} for $j \in [n-1]$
and either: (a). I_n 1-wraps with I_1 or (b). $\dim(I_1) = 1$.
`Add` $[I_1'', \ldots, I_n'']$ to $\mathcal{I}$ such that:
1. for $j \in [2, n]$, $I_j = I_j' \bowtie_1 I_j''$ and $\dim(I_j') = 1$
2. `if` (a) holds `then` also $I_1 = I_1' \bowtie_1 I_1''$ and $\dim(I_1') = 1$
   `else` (i.e., (b) holds) $I_1'' = (\langle out_1, out_1 \rangle)$, where $I_n = \langle in_j, out_j \rangle_{j \in [\dim(I_n)]}$.

Figure 2 Rules of the fixed-point algorithm solving the k-scoped reachability problem.

As an example, consider again the run of Figure 1. Our fixed-point algorithm computes the canonical thread-interface of the first thread as $(\langle q_0, q_3 \rangle, \langle q_5, q_8 \rangle) \bowtie_2 (\langle q_8, q_{10} \rangle, \langle q_{13}, q_{15} \rangle)$ and that of the second thread as $(\langle q_3, q_5 \rangle, \langle q_{10}, q_{13} \rangle) \bowtie_1 (\langle q_{15}, q_{17} \rangle)$. Thus, only fragments of dimension at most 2 are stored (3 is the dimension of the canonical thread-interfaces).

Transition system. The computation of the fixed-point algorithm described above on an n-MPDS M naturally defines a finite-state nondeterministic transition system. The states of the system are the tuples of fragments of thread-interfaces of dimension at most k, and the initial states and the transitions are given by the rules in Figure 2.

Formally, we define the transition system $\mathcal{A}_M^k = (\mathcal{I}_0, \mathcal{I}, \delta)$ where $\mathcal{I}_0 = \{[I, \emptyset, \ldots, \emptyset] \mid I$ is an initial 1-thread-interface of M and $\dim(I) \leq k\}$ is the set of initial states, $\mathcal{I}$ is the set of states, and $\delta \subseteq \mathcal{I} \times \{1, 2\} \times \mathcal{I}$ is the transition relation and contains all tuples (ν, i, ν') such that ν' is obtained from ν by applying the thread-interface progression rule, if $i = 1$, and the round deletion rule, otherwise (i.e., $i = 2$). A *run* of $\mathcal{A}_M^k$ is any sequence $\pi = \nu_0 \xrightarrow{m_1} \nu_1 \xrightarrow{m_2} \ldots \xrightarrow{m_t} \nu_t$ such that ν_0 is initial, $(\nu_{j-1}, m_j, \nu_j) \in \delta$ for every $j \in [t]$, and ν_t is of the form $[(\langle q, q \rangle), \emptyset, \ldots, \emptyset]$ for some M state q.

Correctness. Fix a n-MPDS $M = (Q, q_0, \Gamma, \{(\delta_i^{int}, \delta_i^{push}, \delta_i^{pop})\}_{i \in [n]})$ and $k \in \mathbb{N}$. Given a run π of $\mathcal{A}_M^k$, let $J_1, \ldots, J_m$ be the sequence of thread-interfaces that are used in the application of the thread-interface progression rule along π (transitions labeled with 1) in the ordering they appear in π. Furthermore, we assume that J_i is appended to the j_i component of the state. With $\mathit{Tuple}(\pi)$ we denote the tuple $[I_1, \ldots, I_n]$ that is obtained starting from ν_0 (the first state of π) by iteratively appending for $i = 1, \ldots, n$, J_i to the j_i'th component via $\bowtie_1$ if J_i replaces the $\emptyset$ by the effect of the corresponding transition, and $\bowtie_2$, otherwise.

By Lemma 4, we can show the following:

▶ **Lemma 7.** *$[I_1, \ldots, I_n]$ is the n-tuple of canonical thread-interfaces of a k-scoped computation of M iff there is a run π of $\mathcal{A}_M^k$ such that $\mathit{Tuple}(\pi) = [I_1, \ldots, I_n]$.*

Let $[I_1, \ldots, I_n]$ be the tuple of canonical thread-interfaces of a k-scoped run of M. By following the decomposition of each I_j given by Lemma 4 and then deleting rounds via transitions labeled with 2, it is possible to show that there is a run π of $\mathcal{A}^k_M$ such that $Tuple(\pi) = [I_1, \ldots, I_n]$, which ends in a state $[(\langle out, out\rangle), \emptyset, \ldots, \emptyset]$, where out is the last out-state of I_n.

Therefore, since the set $\mathcal{I}$ computed by the fixed-point algorithm given earlier in this section is also the set of states of $\mathcal{A}^k_M$, by Lemmas 5 and 7 we get:

▶ **Theorem 8.** *Let M be an n-MPDS, q be an M control state, and $k \in \mathbb{N}$. Then, q is reachable in a k-scoped computation of M iff $[(\langle q, q\rangle), \emptyset, \ldots, \emptyset] \in \mathcal{I}$.*

Sequentialization. It is possible to construct a pushdown system (1-MPDS) P^k_M such that the scope-bounded reachability problem on a given n-MPDS M can be reduced to standard reachability on P^k_M. The pushdown system P^k_M is essentially obtained by composing the transition system $\mathcal{A}^k_M$ with the threads M_i such that each transition $(\nu, 1, \nu')$ of $\mathcal{A}^k_M$ involving an i-thread-interface is replaced by a computation of M_i that computes this i-thread-interface followed by a thread-switch.

By Theorem 8, we can show the following:

▶ **Theorem 9.** *Let M be an n-MPDS and $k \in \mathbb{N}$. Then, the k-scoped reachability for M can be reduced to reachability for the pushdown system P^k_M.*

5 Tree-width of bounded scoped multiply nested words

In this section, we show that the set of multiply nested words corresponding to k-scoped runs of any n-MPDS has tree-width bounded by $2kn$.

To each k-scoped computation ρ of an n-MPDS M we associate a labelled graph nw^ρ, called the *multiply nested word* of ρ, as follows. Let $\rho = C_0 \xrightarrow{\sigma_1} C_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_m} C_m$. Then, $nw^\rho = (V, E_L, \{E_h\}_{h\in[n]})$ where $V = \{v_i \mid i \in [0, m]\}$ is the set of vertices of nw^ρ, $E_L = \{(v_{i-1}, v_i) \mid i \in [m]\}$ is the set of all linear edges, and E_h is the set of all edges (v_i, v_j) such that $\mu^\rho_h(i, j)$ holds true. Figure 3 shows the multi-nested word of the run of Figure 1.

▶ **Definition 10** (TREE-WIDTH)**.** A *tree-decomposition* of a graph $(V, E_1, \ldots, E_m)$ is (T, bag), where T is a binary tree with set of nodes N, and $bag : N \to 2^V$ s.t.

- For every $v \in V$, there is a node $n \in N$ such that $v \in bag(n)$,
- For every $(u, v) \in \bigcup_{i\in[m]} E_i$, there is a node $n \in N$ such that $u, v \in bag(n)$,
- If $u \in bag(n)$ and $u \in bag(n')$, for nodes $n, n' \in N$, then for every n'' that lies on the unique path connecting n and n', $u \in bag(n'')$.

The *width* of a tree decomposition (T, bag) is the size of the largest bag in it, minus one; i.e. $max_{n\in N}\{|bag(n)|\} - 1$. The *tree-width* of a graph is the *smallest* of the widths of any of its tree decompositions. ◀

The tree-width of bounded-scoped multiply nested words. We show that, for any k-scoped computation ρ of an n-MPDS M, the tree-width of the corresponding multiply nested words nw^ρ is bounded by $2nk$. For each nw^ρ, we describe a tree decomposition that uses as basic blocks the tree decompositions of the subgraphs corresponding to thread-interfaces and arrange

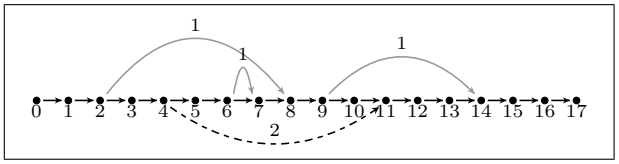

Figure 3 The 2-nested word of the run of Fig. 1.

them into a tree decomposition for the entire graph according to the corresponding run of the transition system $\mathcal{A}^k_M$ (defined in Section 4).

For the rest of the section, fix an n-MPDS M, a k-scoped run ρ of M and a run $\pi = \nu_1 \overset{m_1}{\longmapsto} \nu_2 \overset{m_2}{\longmapsto} \dots \overset{m_{t-1}}{\longmapsto} \nu_t$ of $\mathcal{A}^k_M$ such that $Tuple(\pi)$ is the tuple of canonical thread-interfaces of ρ. Also, denote $nw^\rho = (V, E_L, \{E_h\}_{h\in[n]})$.

Let J_1 be the 1-thread-interface of ν_1, and for $j \in [2, m]$, $\nu_{i_j-1} \overset{1}{\longmapsto} \nu_{i_j}$ be all the 1-transitions of π (i.e., those related to the application of the thread-interface progression rule) and J_j be the thread-interface there used. For $i \in [m]$, let $\rho^i_1, \dots, \rho^i_{r_i}$ be the portions of ρ that correspond to J_i. Note that, except for the starting and the ending configurations each ρ^i_j is disjoint from each other, and ρ can be constructed by stitching the ρ^i_j, one to another, on the starting and ending configurations in some order.

We define $G^i = (V^i, E^i_L, \{E^i_h\}_{h\in[n]})$ as the subgraph of nw^ρ over the vertices $V^i \subseteq V$ that correspond to the configurations visited in the runs $\rho^i_1, \dots, \rho^i_{r_i}$. Note that, E^i_L contains all the edges of E_L that connect two vertices of V^i, E^i_h is empty for $h \neq i$, $h \in [n]$, and E^i_i contains all the edges of E_i that connect two vertices of V^i. We denote with B^i the subset of V^i containing all the vertices that correspond to the starting and the ending configurations of each ρ^i_j, $j \in [r_i]$.

We observe that all the edges of nw^ρ except for those in G^i, $i \in [m]$, do not connect vertices in $V^i \setminus B^i$, and thus B^i contains all the vertices that connect G^i with the rest of the graph nw^ρ.

Given two graphs $G' = (V', E'_L, \{E'_h\}_{h\in[n]})$ and $G'' = (V'', E''_L, \{E''_h\}_{h\in[n]})$ the union of G' and G'', denoted $G' \cup G''$, is the graph $(V' \cup V'', E'_L \cup E''_L, \{E'_h \cup E''_h\}_{h\in[n]})$.

For all the above, clearly nw^ρ can be seen as the union of G^i for $i \in [m]$.

We recall that any subgraph G of a multi-nested word which corresponds to a thread-interface I of dimension k has a tree-decomposition of width at most $2k + 1$ [20]. In this decomposition, the bag of the root contains exactly the vertices corresponding to the starting and ending configurations of the runs corresponding to I, therefore its size is at most $2k$.

For each G^i, $i \in [m]$, denote with $TD_i = (T_i, bag_i)$ the tree-decomposition of G^i as in [20]. Observe that the bag of the root of each T_i is exactly B^i.

Now, define the sequence $\mathcal{B}_1, \dots, \mathcal{B}_t$ as follows. The element $\mathcal{B}_1$ is the set of vertices B^1. For each $i \in [2, t]$ such that $\nu_i = \nu_{i_j}$, i.e., in π the transition from ν_{i-1} to ν_i is an application of the thread-interface progression rule, we set $\mathcal{B}_i = \mathcal{B}_{i-1} \cup B^j$. Otherwise, i.e., the transition from ν_{i-1} to ν_i is an application of the round-deletion rule, we set $\mathcal{B}_i = \mathcal{B}_{i-1} \setminus D^i$, where with D^i we denote the vertices of $\mathcal{B}_{i-1}$ which correspond to the elements that are deleted from the fragments of thread-interface moving from ν_{i-1} to ν_i in π. Note that for $i \in [m]$, $|\mathcal{B}_i| \leq n(k + 2)$.

A tree-decomposition $TD = (T, bag)$ for nw^ρ is thus as follows (see Figure 4). The leftmost path of T corresponds to the sequence $\nu_1\nu_{i_2} \dots \nu_{i_m}$. Precisely, denoting with $u_1 \dots u_m$ the leftmost path of T, $bag(u_1) = \mathcal{B}_1$, and for $j \in [2, m]$, $bag(u_j) = \mathcal{B}_{i_j}$.

By the definition of $\mathcal{A}^k_M$, if a vertex v is in the bag of two nodes u_i and u_j, $i \leq j$, then v is also in all the bags of the nodes of the path $u_i u_{i+1} \dots u_j$.

The rest of TD is given by adding TD_j as right child of the nodes u_j, $j \in [m]$.

Recall that the edges outside of G^i cannot have as an end-point a vertex of G^i which is not in B^i.

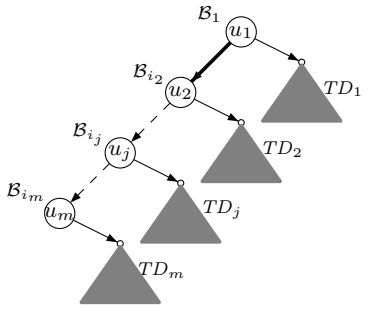

Figure 4 Tree-decomposition TD.

Moreover, we have that if a vertex v is in the bags
of two nodes on the leftmost path, then it is also in bags of all the nodes in the between. Therefore, since each subtree TD_i is a tree-decomposition and the bag of the root of each T_i is exactly B^i, we can conclude that TD is a tree decomposition for nw^ρ. Moreover, since TD_i has tree-width at most $2k$ and $|bag(u_i)| \leq n(k+2)$, for $i \in [m]$, we get that the tree-width of nw^ρ is at most $2kn$.

▶ **Lemma 11.** *For any $k, n \in \mathbb{N}$, the class of all k-scoped n-nested word graphs has tree-width bounded by $2nk$.*

Multiply nested word graphs are Monadic Second Order (MSO) definable (see [20]). Furthermore, the bounded scope restriction is easily expressible in MSO on multiply nested words. Thus, following the approach of [20] we have.

▶ **Theorem 12.** *The satisfiability problem of any MSO sentence on the class of all k-scoped n-nested word graphs is decidable.*

6 Conclusions and Future Work

We have presented a new algorithm for solving the reachability problem on scope-bounded Mpds. Our solution is fixed-point and allows a new sequentialization algorithm, which is useful for the analysis of concurrent programs by means of sequential verification tools. We have also shown that the class of multiply nested words for scope-bounded executions has bounded tree-width. Below we describe possible implications and further explorations that we believe it is worth to pursue.

Our fixed-point formulation for the reachability problem of scope-bounded Mpds has direct encoding in Getafix [11], an efficient verification tool for sequential and concurrent Boolean programs. It would be interesting to empirically evaluate our solution in Getafix on several abstractions of device drivers.

The sequentialization we propose can be extended to real programming languages and can be realised as a code-to-code translation from concurrent to sequential programs. We plan to implement this sequentialization for the C language by using the frama-C framework, and employ several sequential verification tools for the analysis, such as Corral [17] which has been optimized for sequentializations of concurrent programs.

Recently, Madhusudan and Parlato have shown that the reachability problem of several restrictions of Mpds is decidable by providing a uniform decidability schema [20]. In this paper we show that also the scope-bounded restriction fits in this framework. Furthermore, in [20], it is shown the decidability smoothly extends to any MSO property as well as to infinite runs. This allows us to get a series of new decidability results for scope-bounded Mpds, such as the decidability of Linear Temporal Logic and the concurrent temporal logic introduced in [16].

References

1 Mohamed Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. Linear-time model-checking for multithreaded programs under scope-bounding. In *ATVA*, volume 7561 of *LNCS*, pages 152–166, 2012.

2 Mohamed Faouzi Atig, Ahmed Bouajjani, and Gennaro Parlato. Getting rid of store-buffers in TSO analysis. In *CAV*, volume 6806 of *LNCS*, pages 99–115. Springer, 2011.

3 Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. In *TACAS*, volume 5505 of *LNCS*, pages 107–123. Springer, 2009.

4 Ahmed Bouajjani and Michael Emmi. Bounded phase analysis of message-passing programs. In *TACAS*, volume 7214 of *LNCS*, pages 451–465. Springer, 2012.

5 Ahmed Bouajjani, Michael Emmi, and Gennaro Parlato. On sequentializing concurrent programs. In *SAS*, volume 6887 of *LNCS*, pages 129–145. Springer, 2011.

6 Ahmed Bouajjani, Séverine Fratani, and Shaz Qadeer. Context-bounded analysis of multithreaded programs with dynamic linked structures. In *CAV*, volume 4590 of *LNCS*, pages 207–220. Springer, 2007.

7 Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. Mso decidability of multi-pushdown systems via split-width. In *CONCUR*, volume 7454 of *LNCS*, pages 547–561. Springer, 2012.

8 Michael Emmi, Shaz Qadeer, and Zvonimir Rakamaric. Delay-bounded scheduling. In *ACM SIGPLAN-SIGACT POPL*, pages 411–422, 2011.

9 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE Computer Society, 2007.

10 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. An infinite automaton characterization of double exponential time. In *CSL*, volume 5213 of *LNCS*, pages 33–48. Springer, 2008.

11 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Analyzing recursive programs using a fixed-point calculus. In *PLDI*, pages 211–222. ACM, 2009.

12 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Reducing context-bounded concurrent reachability to sequential reachability. In *CAV*, volume 5643 of *LNCS*, pages 477–492. Springer, 2009.

13 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV*, volume 6174 of *LNCS*, pages 629–644. Springer, 2010.

14 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Sequentializing parameterized programs. In *FIT*, volume 87 of *EPTCS*, pages 34–47, 2012.

15 Salvatore La Torre and Margherita Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR*, volume 6901 of *LNCS*, pages 203–218. Springer, 2011.

16 Salvatore La Torre and Margherita Napoli. A temporal logic for multi-threaded programs. In *IFIP TCS*, volume 7604 of *LNCS*, pages 225–239. Springer, 2012.

17 Akash Lal, Shaz Qadeer, and Shuvendu K. Lahiri. A solver for reachability modulo theories. In *CAV*, volume 7358 of *LNCS*, pages 427–443. Springer, 2012.

18 Akash Lal and Thomas W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. In *CAV*, volume 5123 of *LNCS*, pages 37–51. Springer, 2008.

19 Akash Lal, Tayssir Touili, Nicholas Kidd, and Thomas W. Reps. Interprocedural analysis of concurrent programs under a context bound. In *TACAS*, volume 4963 of *LNCS*, pages 282–298. Springer, 2008.

20 P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In *ACM SIGPLAN-SIGACT POPL*, pages 283–294, 2011.

21 Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *TACAS*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

22 Dejvuth Suwimonteerabuth, Javier Esparza, and Stefan Schwoon. Symbolic context-bounded analysis of multithreaded Java programs. In *SPIN*, volume 5156 of *LNCS*, pages 270–287. Springer, 2008.